МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ ТОЧНОЙ МЕХАНИКИ И ОПТИКИ (ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)

Факультет Информационных технологий и программирования

Направление (специальность) Прикладная математика и информатика

Квалификация (степень) Бакалавр

Специализация

Кафедра Компьютерных технологий Группа 439

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Игровой метод тестирования решений на соревнованиях по программированию и его реализация

Автор квалификационной работы Корнеев Г.А.
(Фамилия, И., О.) (подпись)

Руководитель Парфенов В.Г.
(Фамилия, И., О.) (подпись)

К о н с у л ь т а н т ы :

а) По экономике и организации производства
(Фамилия, И., О.) (подпись)

б) По безопасности жизнедеятельности и экологии
(Фамилия, И., О.) (подпись)

в)
(Фамилия, И., О.) (подпись)

К з а щ и т е д о п у с т и т ь

Зав. кафедрой Васильев В.Н.
(Фамилия, И., О.) (подпись)

“___”_____ 2002 г.

Санкт-Петербург, 2002 г.

Квалификационная работа выполнена с оценкой_____

Дата защиты “____”_____2002 г.

Секретарь ГАК_____

Листов хранения_____

Чертежей хранения_____

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Основной целью данной работы является описание нового подхода к автоматическому тестированию программ. Где применяется подобное тестирование?

В настоящее время широкое распространение получили различные соревнования по программированию, например, этапы Командного студенческого чемпионата мира по программированию (ACM ICPC), Всероссийские командные олимпиады школьников по программированию. Кроме того, указанные задачи встречаются в современных технологиях дистанционного обучения, в частности, в проекте Интернет–школы программирования (Internet Programming School, IPS), реализуемом на кафедре компьютерных технологий СПбГИТМО (ТУ).

Так же в данной работе разрабатывается новый подход к тестированию задач, основанный на игровой стратегии. Это означает, что вместо традиционного тестирования на заранее определенном наборе тестов наша система “играет” с проверяемой программой, и по результатам данной “игры” определяет общий результат. Данный метод позволяет адекватно оценивать задачи, для которых оптимальное решение практически не достижимо в силу высоких требований к вычислительным ресурсам. Кроме того, позволяет тестировать управляющие программы, когда автоматическая тестирующая система эмулирует управляемый объект и оценивает качество управления.

Наконец, универсальность нового подхода позволяет настраивать систему проверки решений задач на работу в режимах тестирования, регламентируемых различными требованиями, которые определяются согласно правилам конкретного соревнования. Например, весьма различаются условия проверки результатов, принятые в упомянутых выше соревнованиях, но и в личных соревнованиях школьников России, а также в соревнованиях, где участники сдают не программы, а только результаты их работы.

Естественно, формулировка первой задачи – разработки нового подхода к тестированию – влечет необходимость его реализации в рамках программного комплекса. Указанный комплекс реализован в ядре автоматической тестирующей системы PCMS2 Kernel и успешно прошел неоднократную проверку в соревнованиях по программированию, а также в качестве проверяющего модуля системы дистанционного обучения. В настоящее время наша разработка обеспечивает функционирование IPS в части тестирования обучаемых, а также находит применение в учебном процессе на кафедре КТ СПбГИТМО (ТУ).

Первая глава работы посвящена обзору известных систем автоматического тестирования. Там же обсуждаются проблемы, встающие перед разработчиками подобных систем.

Во второй главе, на основании анализа различных систем автоматического тестирования, а также опыта проведения соревнований, сформулированы основные принципы организации системы. Подробно рассматривается игровая стратегия тестирования решений. Далее приведено общее описание и модель ядра автоматической системы тестирования PCMS2 Kernel. Кроме того, уделено внимание сравнительному анализу сходства и отличия в режимах тестирования задач, определяемых различными регламентами.

В третьей главе приведена подробная архитектура PCMS2 Kernel, позволяющая распараллелить тестирование решений на нескольких компьютерах. Подробно описаны пользовательские интерфейсы и взаимодействие различных частей системы автоматического тестирования.

В заключении дано описание текущего состояния разработки и перспективы ее развития.

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ

В настоящее время, автоматическое тестирование программных решений, обсуждается достаточно широко. На данный момент, разработаны и успешно используются различные методики автоматического тестирования и программные продукты, основанные на их применении. Но большинство таких продуктов “заточены” под проведение соревнований с конкретным регламентом и практически не подлежат изменению для поддержки соревнований с существенно отличающимися регламентами проведения.

Так же, в последнее время, все большую популярность приобретают соревнования реального времени, вытесняющие классические соревнования с отложенной проверкой. Но соревнования такого рода требуют значительно более высокого уровня организации проверки решений участников, так как время, выделяемое на тестирование решения, измеряется минутами. Напротив, в соревнованиях с отложенной проверкой время тестирования решения практически не ограничено.

1.1. Термины и понятия

В данном разделе описаны термины, используемые других частях представленной работы. При этом смысл многих терминов сужен, по сравнению, с их обычным смыслом. Это связано, с тем, что данная работа ориентирована в первую очередь, на тестирование задач на соревнованиях по программированию. В дальнейшем приведенные термины будут использоваться в указанных значениях, если не оговорено обратное.

1.1.1. Автоматическое тестирование

Автоматическое тестирование (Automated judging) — новая концепция подхода к тестированию задач, основанная на полностью автоматизированном выполнении всех действий, необходимых для проверки решения, сданного участником соревнования, не требующем, но и не исключающем вмешательство

человека. В частности, автоматическое тестирование может происходить круглосуточно, в соответствии с запросами пользователей системы автоматического тестирования, реализующей данный подход.

Автоматическая система тестирования (Automated judging system) — комплекс программно-аппаратных средств, реализующих концепцию автоматического тестирования.

Задача (Problem) — Не формальное, но полное описание зависимости выходных данных от входных данных. Так же в условии задачи, обычно указывается максимальное время работы правильного решения на одном тесте, и максимальное количество памяти, которое может использовать решение участника.

Подход, решение участника (Run) — попытка участника сдать задачу, с целью получить положительный результат проверки. Автоматическая система тестирования должна проверить решение участника, и сообщить ему полученный результат. В случае соревнований реального времени, так же обновляется таблица текущих результатов.

Тест (Test) — единица тестирования задачи, в большинстве автоматических систем тестирования. Содержание теста определятся до процесса тестирования. Набор тестов для каждой задачи не изменен в течение всего соревнования. Тест может быть пройденным (в случае, если, решение участника не выполнило недопустимых операций, и выданный результат корректен), и не пройденным, в противном случае.

Проверяющая программа (Tester) — программа, определяющая правильность результата, выданного программой участника на некотором тесте. Обычно на вход проверяющей программы подаются несколько файлов: входной файл, полученный решением участника, выходной файл — результат, сформированный решением, а так же, файл, содержащий информацию помогающую проверить данный тест (если ответ задачи единственен, данный файл содержит его).

Результат проверки (Outcome) — общий результат проверки задачи. Зависит от регламента проведения соревнований, может быть как простым: зачтено / не зачтено, так и более сложным, например, количество очков, полученных участником за данную задачу, или содержать в себе причину, вызвавшую отказ в зачтении задачи.

Администратор — лицо (или группа лиц), ответственное за поддержание системы автоматического тестирования в рабочем состоянии, добавление задач и другие работы по ее обслуживанию.

1.1.2. Соревнования по программированию

Соревнование (Contest) — в рамках данной работы, соревнование определяется правилами его проведения (регламентом), набором задач, доступных для решения, временем и местом проведения соревнования. Результатом соревнования является итоговая таблица, характеризующая рейтинг участников, по определенной шкале.

Регламент соревнования (Contest rules) — порядок регистрации участников, проведения соревнования, правила подсчета результатов в целом, и по каждой задаче в отдельности, допустимые к использованию языки и средства программирования. В данной работе регламент соревнования рассматривается только в части, относящейся к правилам оценки одной задачи, так как задача является базовым объектом тестирования решения участника, в частности, набор языков программирования, допустимых для ее решения. Описание различных регламентов проведения соревнований можно найти в [4] и [5], а так же в других разделах данной работы.

Участник, Команда (Participant, Team) — Лицо, участвующее в соревновании (группа лиц, участвующих в соревновании как единое целое). Команда характеризуется в таблице результатов одной строкой, в не зависимости, от реального количества лиц ее составляющего. Правила отбора участников, и комплектования команд определяются регламентом соревнования. Так как, для

отдельных членов команды результаты не подводятся, данной работе участники и команды не разделяются, и для их обозначения используется слово “участник”.

Соревнование реального времени — соревнование, по регламенту которого участники могут сдавать решения во время проведения соревнования, и должны получать соответствующие результаты в течение оговоренного периода с момента сдачи решения на проверку. Примером таких соревнований могут служить различные этапы Командного студенческого чемпионата мира по программированию.

Соревнование с отложенной проверкой — соревнования, по регламенту которых решения проверяются по окончанию соревнования. Примером соревнований данного типа являются различные этапы Всероссийской олимпиады школьников по информатике.

Интернет–соревнование — соревнование, проводимое посредством глобальной сети Интернет, при этом, участники соревнования могут находиться в произвольной точке Земли. Специфика данных соревнований допускает весьма значительное количество участников, а также возможность проведения круглосуточных соревнований.

Очное соревнование — соревнование, участники которого непосредственно присутствуют в зоне проведения соревнования. В этом случае их количество ограничено существующей технической базой и известно до проведения соревнования.

Дистанционное соревнование — очное соревнование, одновременно проводимое в глобальной сети Интернет. При этом, участники, использующие Интернет, видят результаты участников очного соревнования, но не наоборот. Данный вид соревнований приобретает все большую популярность, так как для их проведения, практически не требуется дополнительная инфраструктура (по сравнению с очными соревнованиями).

1.1.3. Используемые сокращения

APPES — Automated Programming Problem Evaluation System (Автоматическая система проверки задач по программированию). Предшественник PCMS2 Kernel.

PCMS2 — Programming Contest Management System v. 2 (Система управления соревнованиями по программированию, версия 2).

PCMS2 Kernel — Ядро PCMS2, осуществляющее автоматическую проверку решений участников.

ROI — Russian Olympiad in Informatics (Всероссийская олимпиада школьников по программированию).

IOI — International Olympiad in Informatics (Международная олимпиада школьников по программированию).

IPS — Internet Programming School (Интернет-школа программирования).

ACM ICPC — ACM International Collegiate Programming Contest (Студенческий командный чемпионат мира по программированию).

1.2. Обоснование актуальности создания автоматической проверяющей системы

На данный момент, практически отсутствуют статьи, посвященные системам автоматического тестирования задач, хотя в разных странах созданы и успешно действуют несколько таких систем. Но работать, с большей их частью, не говоря уж о расширении возможностей, могут только авторы.

В то же время, проводится все больше различных соревнований по программированию, на большинстве которых участники, как результат своей работы сдают исходные тексты программ. В дальнейшем эти программы необходимо как-то оценивать.

При небольших масштабах соревнования (участники сдают несколько десятков решений) часто применяются методы ручного тестирования. В этом случае, член жюри самостоятельно компилирует каждое представленное решение,

запускает его на последовательности тестов из заранее определенного (но не обязательно известного участникам!) набора, и, наконец, оценивает результат, выданный программой на каждом из них.

Данный подход сопряжен с большими затратами времени и ручного труда. В частности, как показывает опыт, неверно написанные программы участников часто мешают нормальной работе компьютера проверяющего, вследствие чего, последний вынужден производить частые перезагрузки, что еще больше увеличивает временные затраты. Очевидно, данный подход неприменим при более интенсивной нагрузке.

Еще одним недостатком данного подхода является невозможность корректно оценить время, которое будет потрачено на проверку решения участника (так как “хорошее” решение быстро пройдет все тесты, а “плохое” будет вынуждать производить перезагрузку после каждого запуска). Данный недостаток практически не позволяет использовать описанный подход для проведения соревнований реального времени. Несмотря на описанные недостатки, данный метод до сих пор применяется на соревнованиях низкого уровня, таких как, школьные и районные олимпиады школьников по программированию.

Единственным способом преодоления недостатков описанного подхода является автоматизация различных этапов тестирования решений.

Наиболее часто автоматизируемым этапом, является компиляция решений участников. Но, на практике, автоматизация данного этапа не решает ни одну из проблем, присущих ручному тестированию.

Серьезным шагом является автоматизация запуска решений участников на тестах. Обычно, для этого создается программный комплекс, который ограничивает действия, которые может производить решение участника (обычно такую ограниченную среду называют “песочница” — sandbox). В частности, строго ограничивается время и количество памяти, используемое тестируемой программой. Это позволяет ускорить процесс тестирования в несколько раз, а

главное — существенно более точно оценивать время, необходимое на тестирование одного решения участника, что позволяет использовать такую тестирующую систему в соревнованиях реального времени. Такая автоматизация позволяет проверять за разумное время, уже сотни, а не десятки решений.

Тестирующие системы данного типа успешно используются даже на соревнованиях такого уровня, как финал Командного студенческого чемпионата мира по программированию. Но, так как проверка правильности результата, выданного решением участника на тесте, выполняется вручную, то данному методу так же присущи существенные недостатки.

Одним из них является практически полная невозможность проверки задач, с неоднозначным решением, в то же время, задачи такого типа часто встречаются на практике. Нередко, для обеспечения единственности решения, в задачу вносятся определенные ограничения, например, полученное решение, должно быть первым в лексикографическом порядке. Данные ограничения обычно не влияют на алгоритмическую сложность задачи, но зачастую существенно увеличивают ее техническую сложность, что, очевидно, является негативным фактором.

Вторым недостатком частичной автоматизации является необходимость присутствия достаточно квалифицированного человека – а во многих случаях, и группы людей – для выполнения неавтоматизированных этапов, что практически не позволяет создавать системы, работающие круглосуточно. При этом работа, выполняемая человеком, весьма рутинна, из-за чего со временем существенно снижается производительность труда и надежность работы исполнителя. Таким образом, возможности использования частично автоматизированных проверяющих систем ограничены проведением соревнований продолжительностью не превышающих несколько часов. Так же практически исключено проведение дистанционных и интернет-соревнований, как следствие большого числа запросов на проверку.

Итак, чтобы исключить ошибки, более точно — для того, чтобы все участники были в равных условиях, а также для создания тестирующих систем, рассчитанных на круглосуточное использование, необходима полная автоматизация процесса тестирования. В частности, роль администратора такой системы сводится к добавлению новых задач, доступных для тестирования, и, возможно, расширению доступных вычислительных ресурсов.

При этом автоматические системы тестирования могут быть применены во всех типах соревнований, в том числе, для круглосуточных соревнований реального времени. Они могут тестировать тысячи решений в час, что существенно для проведения интернет-соревнований.

Еще одним плюсом автоматических систем тестирования является возможность тестирования задач с существенно не единственным решением, а так же оптимизационных задач. Более того, при использовании игровой стратегии, возможно тестирование задач, для которых точное решение не известно, что существенно увеличивает круг возможных задач.

Кроме того, полная автоматизация системы тестирования позволят тестировать существенно большее количество программ участников за отведенное время. При проведении реальных интернет–соревнований системе иногда приходится тестировать десятки программ в минуту, что, очевидно, невыполнимо ни при ручном, ни при частично автоматизированном тестировании.

Одной из наших целей является создание автоматической тестирующей системы, предназначенной как для проведения соревнований по программированию различного уровня, в том числе, четверть- и полуфинала командного чемпионата мира по программированию, так и для поддержки Интернет–школы программирования, подробно описанной в работе [11]. В частности, необходимо обеспечить возможность круглосуточной работы системы,

без участия администратора; кроме того, – одновременное проведение одного соревнования в нескольких городах и глобальной сети Интернет.

1.3. Обзор существующих систем

Рассмотрим примеры существующих систем автоматического тестирования, успешно используемых для проведения различных соревнований.

Такие системы, распадаются на два практически не пересекающихся класса — системы проведения интернет-соревнований и системы проведения очных соревнований. Существенное отличие первых от вторых заключается в том, что они работают круглосуточно, практически, без вмешательства администратора, а системы проведения очных соревнований используются только непродолжительный период времени, во время проведения соревнования как такового, и постоянно находятся под наблюдением администратора.

1.3.1. Системы проведения интернет-соревнований

Рассмотрим основных представителей класса систем проведения интернет-соревнований и архивов задач с автоматической проверкой. Такие системы уже существуют как в Росси, так и за рубежом.

Существенным недостатком дистанционных соревнований является их временная “привязка” к очным соревнованиям, что практически исключает возможность участия лиц, разница по времени с которыми превышает 8-10 часов. В то же время, дистанционные олимпиады обычно проводятся вместе с очными соревнованиями достаточно высокого уровня и участвовать в них не подготовленным участникам не интересно, так как предлагаемые задачи слишком сложны. Системы проведения интеренет-соревнований обычно позволяют не только участвовать в различных дистанционных соревнованиях, но и создавать собственные соревнования, необходимой сложности и проводимые в требуемое время, что существенно повышает интерес к ним. Так же для них имеется возможность участвовать в одном соревновании несколько раз, с некоторым

временным интервалом, что позволяет участникам сравнивать результаты своих выступлений и отслеживать свой прогресс. Опыт показывает, что многие участники теоретически знают, как решать задачу, но не могут корректно реализовать имеющийся алгоритм на выбранном языке программирования. Таким образом, наравне с оценкой алгоритмов участники имеют возможность проверить качество кодирования.

Системы проведения интернет-соревнований требуют постоянной поддержки и пополнения набора задач, доступных для тестирования. По этому, системы автоматического тестирования данного класса мало распространены. Фактически эффективно поддерживать такую систему могу только организации с развитой инфраструктурой проведения соревнований по программированию, имеющие солидную базу для подготовки победителей соревнований самого высокого уровня. Только несколько российских и зарубежных ВУЗов удовлетворяют данным требованиям, что существенно сужает круг претендентов на создание такой системы.

Еще одной проблемой является написание надежной системы автоматического тестирования, которая будет успешно сопротивляться попыткам нарушить ее работу. Разрешение данной проблемы самой по себе является весьма не тривиальной задачей.

Рассмотрим подробнее некоторых представителей систем проведения интеренет-соревнований.

Valladolid Programming Contest Problem Set [6] — Старейший архив олимпиадных задач. На данный момент, является одним из самых популярных сервисов такого рода в Интернет. Содержит сотни (!) задач с тестами. К сожалению, все общение с тестирующей системой происходит по e-mail, что является существенным недостатком для некоторых пользователей. Еще одним недостатком данной системы является использование только операционной системы Linux, и соответствующих языков программирования (GNU C, GNU C++, Free Pascal), что существенно сужает количество возможных участников. Для

каждой задачи подводится статистика — рейтинг сложности (отношение количества правильных решений задачи к общему количеству попыток сдать данную задачу), что позволяет участникам выбирать посильные для них задачи. Архив активно пополняется задачами с различных соревнований, как очных, так и проводимых через Интернет.

Ural State University Problem Set with Online Judge System [7] — Архив задач и система проведения дистанционных олимпиад. Данная система популярна в России и Китае. Дистанционные олимпиады обычно проводятся одновременно с очными олимпиадами Уральского Государственного Университета. О проведении соревнования участники предупреждаются по e-mail, примерно за неделю до его начала. Обычно в соревнованиях такого рода участвуют 200-300 команд. Для подсчета результатов используются правила ACM ICPC. Для участников доступен стандартный набор языков программирования: C, C++ и Pascal (Delphi). К сожалению, данная система автоматического тестирования допускает только тестирование зада с однозначным ответом (выходные данные, сформированные программой участника, сравниваются с эталонными).

1.3.2. Системы проведения очных соревнований

Системы проведения очных соревнований специально предназначены для очных олимпиад и не включают возможности проведения даже дистанционных соревнований.

Рассмотрим конкретные примеры автоматических проверяющих систем этого класса.

NPC2 (Антон Суханов, Роман Елизаров) — Одна из первых систем автоматического тестирования. Предназначена для проведения соревнований, с использованием сети Novel Netware, позволяет тестировать только программы написанные под DOS. Данное ограничение является существенным, и для его устранения необходимо переписывать тестирующую систему заново. При проверке, нагрузка может быть распределена на несколько компьютеров,

связанных в локальной сетью. Данная автоматическая система тестирования была успешно использована для проведения многочисленных соревнований по программированию. Для написания проверяющих программ для этой системы была разработана библиотека Testlib, различные модификации которой, в данное время, используются практически во всех российских системах автоматического тестирования. Исходно, система предназначалась для проведения соревнований по регламенту ACM ICPC, в последствие она была адоптирована для проведения Всероссийских Олимпиад Школьников по Информатике.

Cyber Judge (Максим Бабенко) — Относительно новая система автоматического тестирования, предназначенная для проведения соревнований по регламенту Всероссийских олимпиад школьников по информатике. Особенностью данной системы является возможность не только автоматического, но и ручного тестирования решений, что является существенным для ROI, так как по правилам, тестирование происходит в присутствии участников. Данная система поддерживает проверяющие программы, написанные при помощи Testlib, но не совместима с NPC2 по формату конфигурационных файлов. Таким образом, перенос задач из одной системы в другую, сводится к изменению конфигурационных файлов задачи. Данная система находится в стадии модификации, и в будущем, возможно, будет поддерживать проведение дистанционных соревнований.

1.4. Технический и организационный контекст

Рассмотрим структуру разработки системы автоматического тестирования. Автор системы является основным разработчиком ядра тестирующей системы (PCMS2 Kernel) и активно взаимодействует с разработчиками остальных частей. Основным заказчиком является Организационный комитет полуфинала Командного студенческого чемпионата мира по программированию, в лице Парфенова Владимира Глебовича (см. рисунки 1 и 2). Второй областью применения системы автоматического тестирования является автоматизация

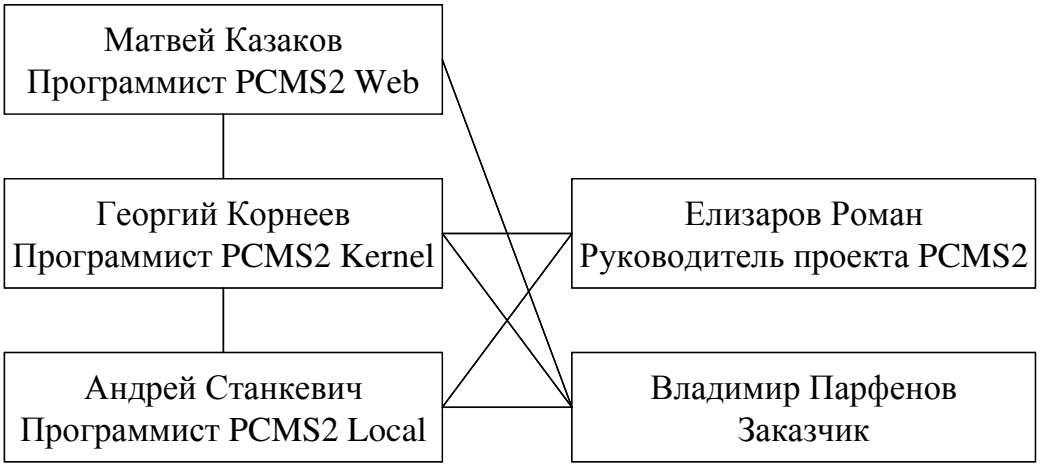

Рис. 1. Схема разработки PCMS2

проверки задач для Интернет-Школы Программирования и системы организации интернет-соревнований.

Ядро тестирующей системы создавалась как автономный проект и для своего функционирования не требует наличия других частей системы.

Обязанности автора работы включают в себя следующие части:

Разработка идей, лежащих в основе ядра автоматической системы тестирования.

Спецификация интерфейсов тестирующего ядра с другими частями системы.

- Реализация прототипа и работающей версии тестирующего ядра.
- Реализация общей среды исполнения компонент автоматической системы тестирования.
- Реализация бизнес-логики ядра автоматической системы тестирования.
- К тестирующей системе заказчиком были предъявлены следующие требования:
- Эффективная работа на компьютерах уровня Pentium 200 MMX, 64Mb RAM.

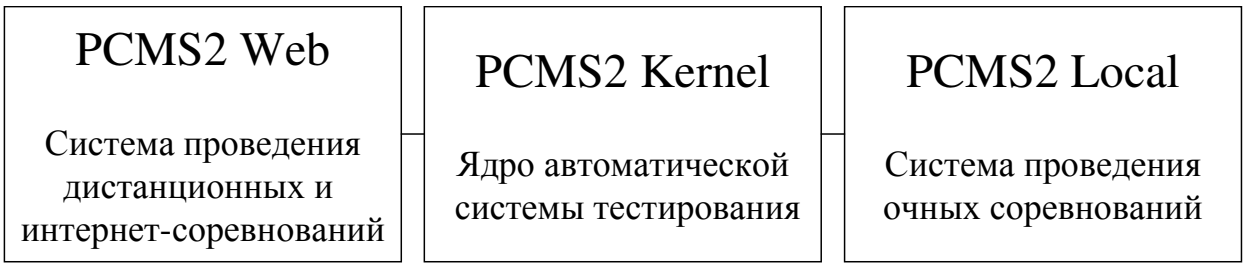

Рис. 2. Общая схема системы автоматического тестирования PCMS2

- Одновременное тестирование программ написанных под DOS, Win16 и Win32.
- Возможность широкого масштабирования проверяющей системы, начиная с работы на одном компьютере. Система должна допускать возможность распределенного тестирования, с использование дополнительных компьютеров, при соответствующем возрастании производительности. При проведении соревнований автоматическая проверяющая система должна эффективно использовать парк из 5-7 компьютеров.
- Автоматическая система тестирования должна допускать круглосуточное использование для применения в проекте IPS.
- Решение участника должно тестироваться не более 5 минут.
- При использовании дополнительных компьютеров система должна одновременно тестировать несколько решений участников.
- В дальнейшем, система должна допускать расширение для одновременного тестирования программ для разных платформ, на пример Linux, путем реализации дополнительных компонентов.

Система должна препятствовать попыткам участников нарушить ее работоспособность.

Фактически, тестирующее ядро разделено на две части — основную (обычно, работающую на одном компьютере, но допускающую распределенную реализацию) и несколько дополнительных компьютеров, на которых запускаются

решения участников, причем вся выход из строя дополнительного компьютера не должен сказываться на работоспособности системы в целом, кроме, может быть, некоторого снижения производительности. При недостаточном количестве ресурсов, решения участников могу запускаться на одном компьютере с основной частью, но это приведет к снижению защищенности системы. Это позволяет производить тестирование с минимальными затратами ресурсов. Для образовательных целей, возможно тестирование даже непосредственно на компьютере участника.

Таким образом, система имеет классическую клиент-серверную архитектуру, которая применяется во многих современных приложениях.

1.5. Уточненные требования к работе

Обобщая вышесказанное, выведем следующие основные цели данной работы:

- Разработать структуру ядра автоматической тестирующей системы, допускающей использование как автономной системы (для поддержки проекта IPS), так и для проведения очных и дистанционных соревнований, с возможностью административного контроля.
- Разработать интерфейсы к тестирующему ядру.
- Интегрировать тестирующее ядро с другими частями автоматической системы тестирования PCMS2.
- Опробование новой схемы тестирования программ, основанной на “игре”.
- Проверка автоматической тестирующей системы на реальных соревнованиях. Анализ результатов ее работы.

Результатом данной работы будет являться работоспособное ядро автоматической системы тестирования, используемое для проведения различных соревнований по программированию, в составе системы автоматического тестирования PCMS2.

ГЛАВА 2. ТЕОРЕТИЧЕСКИЕ РЕЗУЛЬТАТЫ

В данной главе рассматриваются теоретические аспекты создания ядра автоматической системы тестирования. Приведена общая схема тестирующего ядра. Рассмотрены различные регламенты соревнований и их реализация на основе данного ядра. Так же рассматриваются вопросы адаптации ядра к тестированию программ написанных для не Wintel-платформ.

Общая схема тестирующего ядра приведена на рисунке 3. Рассмотрим назначение отдельных частей ядра.

RunRepository (Репозиторий) — основное хранилище информации о решениях участников. Единственный компонент, в котором сохраняется информация между перезапусками ядра. Ни одно действие на решение участника не считается выполненным, пока это не отражено в RunRepository. При этом над одним решением одновременно может выполняться несколько действий. Запись и обновление информации в RunRepository построено на транзакциях, что гарантирует целостность данных в любой момент времени. Данная особенность позволяет безболезненно перезапускать ядро целиком или отдельные его компоненты в любой момент времени. При этом практически не совершается повторных операций.

RunRepository является пассивной (серверной) компонентой и не использует другие компоненты. Наоборот, все остальные компоненты обращаются к нему за необходимой информацией. При этом другим компонентам запрещается хранить информацию, влияющую на проверку программ участников (таким образом, они могут только кэшировать внешние данные необходимые для работы).

Complier (Компилятор) — Компонента компилирующая программы участников. Берет информацию о языке программирования и исходные тексты программы из RunRepository, компилирует их и записывает результат обратно в RunRepository.

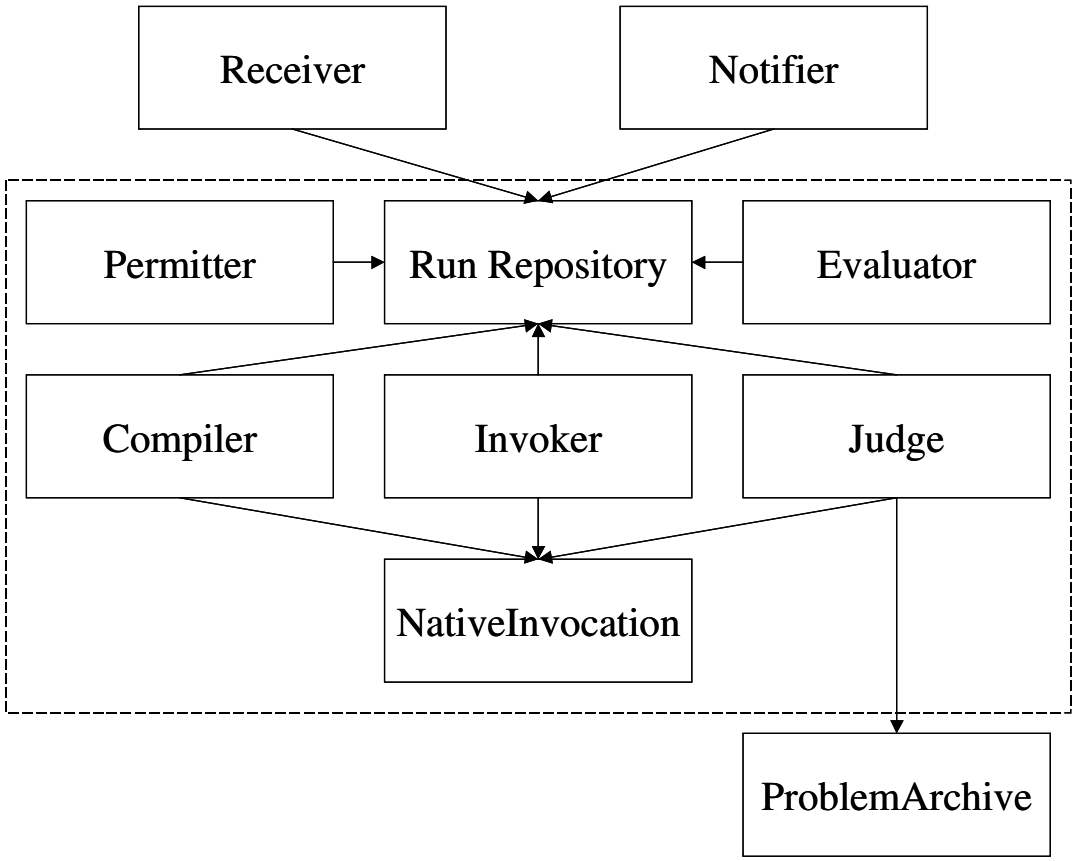

Рис. 3. Схема ядра PCMS2

Judge (Судья) — Подготавливает данные для запуска программы участника, создает запрос на ее исполнение, оценивает результаты, выданные программой участника.

Invoker (Исполнитель) — Исполняет запрос на запуск решения участника. Для исполнения создается песочница (sandbox). Для программы участника контролируются все операции ввода-вывода, использованное процессорное время, количество занятой оперативной памяти и другие параметры. Исполнение решения участника может быть прервано, если программа нарушила правила соревнований, или есть подозрение на то, что программа пытается помешать работе проверяющей системы.

NativeInvocation — Непосредственно исполняет программы на компьютере. Используется для запуска компиляторов, проверяющих программ, решений участников. Получает описание “песочниц” от соответствующих

компонент и следит за их соблюдением. Единственная платформо-зависимая часть ядра системы.

Permitter — Проверяет исходный текст программы на наличие запрещенных конструкций, на пример, ассемблерных вставок и работы с векторами прерываний. Программы, не прошедшие данную проверку дальнейшему тестированию не подлежат.

Evaluator — Вычисляет результат тестирования решения, на основе информации о пройденных тестах. Позволяет использовать сложную логику оценки, когда баллы, получаемые за тест, зависят от результата прохождения других тестов.

Receiver — Принимает решения пользователей и записывает информацию о них в RunRepository. Решение считается не принятым, пока информация о нем не будет успешно записана в репозиторий, что позволяет сообщать пользователю об ошибках, произошедших в процессе отправки решения.

Notifier — Сообщает другим частям системы и участнику результаты тестирования решения. Для одной задачи могут быть использованы несколько различных Notifier. На пример, при проведении интернет-соревнований одновременно используется 2 Notifier’а — один для записи информации в базу данных и последующего формирования таблицы результатов, а второй — для создания e-mail с результатами проверки, который отправляется участнику.

ProblemArchive (Архив задач) — Хранит условия задач и правила их проверки. Проверяющих программы являются частью архива задач. В то же время, предусмотрена возможность использования стандартных проверяющих программ, являющихся частью системы автоматического тестирования.

2.1. Анализ предыдущих разработок

2.1.1. Система автоматического тестирования NPC2

Данная система автоматического тестирования (NPC2 = Network Problem Contest v. 2) была написана Антоном Сухановым и модифицирована Романом Елизаровым в 1998г. Это был один из первых в России опыт написания подобных систем.

NPC2 была существенным шагом вперед, по сравнению с применявшимися в то время системами тестирования и была успешно использована для организации большого количества соревнований в 1998-1999г.

В NPC2 изначально закладывалась возможность работы на нескольких компьютерах, связанных в сеть Novell Netware. При этом один из компьютеров назначался главным (на нем запускался “Next”), а другие — подчиненные (на них запускался “Judge”).

Next занимался распределением приходящих решений для Judge, при необходимости помещая их в очередь. Так же он принимал от Judge результаты проверки решений участников и составлял итоговые таблицы.

Тестирование каждого решения было организовано на одном компьютере, что являлось существенным недостатком данной системы. В тоже время, отказ одного из Judge не приводил к краху всей системы проверки.

Для общения Next и Judge использовались сетевые протоколы Novell Netware и сетевые диски. По этому, теоретически, участники могли написать программу, которая подменяла собой Judge, и сообщала Next, о том, что проверка программы прошла успешно.

Еще одним недостатком NPC2 являлось частое зависание компьютеров, используемых для тестирования, так как корректно прерывать зависшие программы под DOS невозможно. В тоже время, система достаточно успешно справлялась с зависаниями, вызванными не оптимальностью алгоритма или его ошибочной реализации.

Самым же большим недостатком NPC2 являлась существенная ориентация на тестирование DOS-программ. Фактически, систему было невозможно адаптировать к тестированию программ для Windows, а тем более, для других операционных систем.

Так же Антоном Сухановым была написана библиотека (Testlib), позволяющая существенно упростить написание проверяющих программ. В ней были реализованы развитые средства ввода слабоструктурированной информации. Что позволяло сосредоточиться на собственно алгоритме проверки правильности полученного решения, а не на технических сложностях. Расширения данной библиотеки используются практически во всех современных автоматических системах тестирования, таких как Cyber Judge и PCMS2.

В результате использование NPC2 стало очевидно, что программы, запускающие решения участников и оценивающие результаты их работы, должны быть разнесены, так как, в противном случае, решения участников потенциально смогут искажать результаты проверки. Данный опыт был учтен при проектировании системы PCMS2, что выразилось в выделении как самостоятельной части системы, только исполняющей решения участников.

2.1.2. Система автоматического тестирования APPES

APPES (Automated Programming Problems Evaluation System) была реализована на кафедре Компьютерных Технологий СПбГИТМО (ТУ) в 1999-2000г. группой программистов, в которую входил и автор данной работы. На ней были опробованы многие решения, включенные в последствие в PCMS2.

В APPES были предприняты шаги в сторону разделения тестирующего ядра на отдельные независимые компоненты, но эти идеи не были доведены до логического завершения. В результате, полученная система тестирования была достаточно модульной, для использования для тестирования нескольких компьютеров, но основная часть тестирующего ядра была практически

монолитной, хотя его внутренняя архитектура сохранила следы разбиения на части.

В PCMS2 был учтен опыт, полученный при разработке APPES, и в нее была изначально заложена компонентная архитектура, что позволяет, во-первых, сделать систему действительно модульной, а во-вторых, отдельные части системы могут быть написаны разными программистами, без дополнительного согласования. Таким образом, PCMS2 существенно более легко расширяема, чем ее предшественники.

2.2. Автоматическое тестирование

Задача процедуры тестирования программного решения состоит в определении, решает ли тестируемая программа требуемую задачу.

В последнее время широкое применение находят методы, основанные на анализе исходного кода программ. Но для любого достаточно сложного языка программирования (эквивалентного машине Тьюринга), существуют программы, для которых невозможно эффективно определить, не только решают ли они заданную задачу, но даже, закончит ли программа выполнение за конечное время.

Так как, точное определение корректности программы невозможно, то широкое применение находят различные эмпирические методы тестирования.

При этом в зависим ости от структуры задачи используются различные методы тестирования.

Назовем метод тестирования **косвенным**, если для определения корректности ответа, получаемый на наборе входных данных, производится проверка принадлежности его некоторому, заранее вычисленному множеству.

Косвенные методы обычно применяются при тестировании задач с единственным решением, а так же задач, для которых известны эффективные методы получения всех возможных правильных ответов.

Недостаток косвенных методов состоит в том, что для проверки правильности результата мы должны уметь решать исходную задачу, или даже более сложную (в случае, когда решение исходной задачи не единственно).

Назовем метод тестирования **прямым**, если для проверки правильности результаты нет необходимости иметь решение исходной задачи.

Рассмотрим следующую классификацию задач:

1) По количеству правильных ответов.
   A) Задачи с единственным ответом.
   B) Задачи с конечным множеством правильных ответов.
   C) Задачи с бесконечным множеством правильных ответов.
2) По существованию эффективного метода решения
   A) Для задачи известен эффективный метод получения всех ответов.
   B) Для задачи известен эффективный метод получения некоторого ответа.
   C) Для задачи не известны эффективные методы решения.
3) По существованию эффективного метода проверки решения, не включающего решение исходной задачи.
   A) Для задачи известен эффективный метод проверки, не требующий решения исходной задачи.
   B) Такой метод не известен.

Будем обозначать типы задач трехбуквенными сокращениями, где каждая буква является значением соответствующего параметра. На пример, задача типа BCA — имеет конечное множество решений, для которого не известны эффективные методы нахождения хотя бы одного элемента, но существует эффективный метод проверки принадлежности данному множеству.

Очевидно, что не все возможные 18 типов задач имеют смысл. Рассмотрим примеры задач для каждого типа.

AAA. Задача извлечения арифметического квадратного корня из целого числа, являющегося полным квадратом. Для данной задачи существует метод

решения линейный по количеству цифр в исходном числе. Так как ответ единственен, данным методом мы получаем все возможные решения. Посредством возведения полученного решения в квадрат, и сравнения с исходным числом производится эффективная проверка правильности результата, не основанная на решении исходной задачи.

AAB. Задача перемножения двух чисел. Очевидно, задача имеет единственное решение, которое можно получить эффективным методом. При этом единственным разумным методом проверки результата является перемножение исходных чисел.

ABA и ABB. Так, как правильно решение единственно, и мы можем эффективно получить некоторое правильное решение, то мы имеем эффективный метод получения всех возможных решений. Таким образом, не существует задач таких типов.

ACA. Задача вычисления дискретного логарифма, т.е. нахождение такого K, что $B = A^K \bmod N$. Если N — простое число, то результат единственен. А для его проверки нужно просто возвести число A в K-ю степень по модулю N.

ACB. Задача разложения числа на простые множители. Одним из этапов проверки решения является определение простоты числа, которая не может быть произведена эффективно.

BAA. Представление натурального числа N в виде суммы двух различных натуральных слагаемых. Мощность множества решений равна $\lfloor N/2 \rfloor - 1$. Проверка производится простым сложением полученных слагаемых.

BAB. Нахождение минимального пути в графе.

BBA. Нахождение гамильтонова пути в плотном графе.

BBB. Нахождение минимального пути во взвешенном графе.

BCA. Задача разложения составного числа на пару натуральных множителей, не равных единице. Задача имеет конечное множество правильных ответов, т.е. его мощность не превосходит некоторого числа. Но эффективного метода нахождения хотя бы одного решения не известно. При этом, если ответ

найден, проверить его не составляет труда — необходимо просто перемножить полученные множители и сравнить результат с исходным числом.

BCB. Задача о назначениях.

CAA и CAB, так как все элементы бесконечного множества, не могут быть получены, то зада данных типов не существуют.

CBA. Задача нахождения Пифагоровой тройки (натуральных чисел A, B и C, таких, что $A^2 + B^2 = C^2$), все числа которой больше заданного.

CCA. Поиск простого числа, больше заданного.

CCB. Численное решение “Задачи трех тел”

По отношению к тестированию задачи разбиваются на три класса:

типы AAA, AAB, BAA, BAB, ACB допускают прямое тестирование.

типы AAA, ACA, BAA, BBA, BCA, CBA, CCA, допускающие косвенное тестирование.

типы BBB, BCB, CBB, CCB Задачи, не допускающие не прямых не косвенных методов тестирования. Для задач этого класса не существует эффективного метода проверки корректности ответа.

При этом, путем наложения дополнительных условий на ответ, задачи типов BBB и CCB зачастую удается преобразовать в задачи с единственным решением, что переводит их в класс задач, допускающих прямое тестирование.

Рассмотрим различные методы тестирования.

2.2.1. Тестирование на наборе тестов

При использовании данного метода тестирования программа считается корректной, если она выдает правильный результат на некотором конечном подмножестве множества всевозможных входных данных

Данный метод имеет как прямой, так и косвенный вариант, в зависимости, от способа проверки ответа.

В прямом варианте полученный ответ проверяется с помощью соответствующей процедуры, а в косвенном ответ сравнивается со всеми правильными ответами для данного набора входных данных (обычно применяется в случае единственности ответа).

2.2.2. Вероятностные методы тестирования

Программа считается корректной, если она выдает правильные ответы на заданном количестве случайных наборов входных данных.

Данный метод обычно используется как косвенный, в случаях, когда известен эффективный метод проверки правильности решения, но не нахождения решения как такового и может быть применен.

2.2.3. Тестирование посредством эмулирования

Программе на вход подается некоторая исходная ситуация, после чего она изменяется, в соответствии с полученным ответом и вновь подается на вход тестируемой программе. Если после ряда запусков программе удается добиться требуемого результата, она считается корректной.

Обычно используется для программ управления или программ для игр (в математическом смысле этого слова), в которых не известна оптимальная стратегия. В первом случае, тестирующая программа эмулирует управляемый объект, во втором, “играет против” тестируемой программы.

Все приведенные методы тестирования могут быть реализованы в виде “диалога” тестирующей и тестируемой программы, что позволяет реализовать единую тестирующую систему, включающую их. Данный подход и был заложен, как базовый в автоматическую тестирующую систему PCMS2.

2.3. Игровая стратегия тестирования

На данный момент подавляющее большинство систем автоматического тестирования использует по тестовый подход. В рамках данного подхода тестируемая программа запускается на заранее определенном наборе тестов,

после чего выданный ею результат проверяется. Существенным недостатком данного подхода является невозможность адаптации системы тестов под конкретное решение, для определения его сильных и слабых сторон.

Автором данной работы была предложена игровая стратегия тестирования. В ее рамках тестирующая система “играет” с тестируемой программой.

2.3.1. Описание игровой стратегии тестирования

Первый ход осуществляет тестирующая система (проверяющая программа). Она подготавливает входные данные, для запуска тестируемой программы, а так же устанавливает ограничения на количество ресурсов, которые будут доступны ей.

Далее ход переходит к тестируемой программе. Она запускается на входных данных, подготовленных тестирующей системой, и выдает некоторый результат. При этом осуществляется контроль за объемом использованных ресурсов.

Ход вновь получает тестирующая система. Она анализирует результат, полученный тестируемой программой. На основе результатов анализа принимается решение о продолжении тестирования.

Если тестирование продолжается, то проверяющая система подготавливает новый набор входных данных, при этом может быть использована информация, полученная от тестируемой программы, на предыдущих шагах. После чего ход снова передается тестируемой программе, и так далее.

2.3.2. Преимущества игровой стратегии тестирования

Игровая стратегия тестирования позволяет повысить гибкость проверки решений и существенно расширяет круг задач, доступных для тестирования.

Очевидно, по тестовый подход является подмножеством предложенной стратегии тестирования. При этом, информация, полученная при запуске программы участника на одном тесте, ни как не используется для проверки других.

То, что, в свой ход, автоматическая система тестирования может производить произвольные действия (в том числе, и запуск внешних программ), позволяет без дополнительных механизмов осуществлять вероятностное тестирование. При этом, тестирующая система, создает случайный набор входных данных, удовлетворяющих условию задачи и передает ход тестируемой программе. Для проверки полученных результатов возможны следующие подходы:

- Тестирующая программа непосредственно проверяет правильность полученного ответа. Применяется при тестировании задач классов ??A.
- Запускается правильное решение (решение жюри). Результаты, выданные тестируемой программой и правильным решением сравниваются. Применяется для тестирования задач с единственным правильным решением.
- Запускается генератор множества всех правильных решений. Результат, выданный тестируемой программой, проверяется на принадлежность сгенерированному множеству. Применяется для тестирования задач класса ?A?, при не большой верхней границе мощности множества правильных ответов.
- Запускается решение жюри и полученный результат, сравнивается по оптимальности с результатом, выданным тестируемой программой. Данный подход используется для тестирования оптимизационных задач, для которых точный ответ получить не представляется возможным (на пример, задачи Коммивояжера для плотных графов с большим числом вершин).

Использования последнего из предложенных подходов, для тестирования на заранее заданном наборе тестов, позволяет осуществлять детерминированное тестирование, задач, точное решение для которых не известно.

Дополнительной возможностью игровой стратегии тестирования является сравнительное тестирование игровых задач. При этом проверяющая программа действительно играет с тестируемой. В начале, задается исходная позиция в игре, после чего, одна из сторон делает первый ход и передает ход другой стороне, которая в свою очередь делает свой ход и т.д. При этом проверяющая программа проверят корректность производимых ходов обеих сторон, и оценивает результаты игры.

Так же игровая стратегия тестирования позволяет осуществлять проверку программ управления. При этом, проверяющая система, в свой ход, проверяет корректность управляющих воздействий, предложенных тестируемой программой и рассчитывает отклик управляемой системы на произведенные воздействия, после чего подает полученный результат на вход тестируемой программе и т.д. Тестируема программа, признается корректной, при достижении требуемого состояния управляемой системы.

Таким образом, в рамках игрового подхода к тестированию, может быть осуществлен двухуровневый подход к тестированию. На верхнем уровне используется заранее заданный набор тестов. На нижнем уровне производится несколько запусков тестируемой программы, с использованием схемы сравнительного тестирования или тестирования программ управления. На верхнем же уровне, оценивается результат произведенной последовательности запусков в целом (результат игры, состояние управляемой системы, и т.п.).

Описанные в данном разделе новые методы тестирования не могут быть реализованы в рамках классической по тестовой стратегии.

2.3.3. Реализация игровой стратегии тестирования

Для реализации игровой стратегии тестирования автором была предложена следующая схема (рисунок 4):

Подготовка данных соответствует первому ходу тестирующей системы. На данном этапе, ни какой информации о тестируемой программе еще не известно.

Рис. 4. Реализация игровой стратегии тестирования

Далее, в соответствии с игровой стратегией тестирования, производится запуск тестируемой программы.

Далее, ход снова переходит к тестирующей системе, которая осуществляет анализ результатов исполнения тестируемой программы и принимает решение о продолжении тестирования.

Следующие этапы подготовки данных могут активно использовать, информацию, полученную на предыдущих этапах.

2.4. Регламенты соревнований по программированию

В последнее время соревнования по программированию нашли существенное распространение. При этом практически для каждого соревнования разрабатываются новые правила. Таким образом, написать систему автоматического тестирования, подходящую для проведения любых соревнований практически невозможно. Приходится изначально ориентироваться на нескольких поддерживаемых регламентах, при этом, оставляя возможность простого включения в систему других.

Практически все регламенты содержат общую часть — причины, по которым задача (или тест) может быть не зачтена. Приведем перечень данных причин в нотации, принятой на Командном студенческом чемпионате мира по программированию.

Наименование	**Номер теста**	**Комментарий**
Accepted	Нет	Задача была успешно зачтена
Compilation error	Нет	Автоматической проверяющей системе не удалось скомпилировать решение участника
Presentation error	Да	Проверяющей программе не удалось прочитать результат, выведенный программой участника (участник не соблюдает формат, описанный в условии задачи)
Wrong answer	Да	На данном тесте программа участника выдала неправильный ответ
Runtime error	Да	На данном тесте программа участника завершилась с ненулевым кодом ошибки
Crash	Да	На данном тесте программа участника была завершена операционной системой, вследствие выполнения недопустимых действий
Time limit exceeded	Да	На данном тесте программа участника превысила предел времени выполнения, указанный в условии задачи
Memory limit exceeded	Да	На данном тесте программа участника затребовала для работы больше памяти, чем указанно в условии задачи
Security violation	Да	На данном тесте программа участника нарушила правила проведения соревнований,

		или была зафиксирована попытка нарушить функционирование проверяющей системы.

Рассмотрим несколько конкретных, широко распространенных, регламентов соревнований, поддержка которых была включена в автоматическую тестирующую систему PCMS2.

2.4.1. Командный студенческий чемпионат мира по программированию (ACM ICPC)

Данный регламент проведения соревнований является одним из наиболее распространенных. Он используется для проведения как внутрироссиских, так и международных соревнований.

Участники могут отправлять решения на проверку и получать соответствующие результаты в течение всего соревнования.

По правилам данных соревнований задача считается зачтенной, тогда и только тогда, когда представленное решение выдало корректные ответы на все тесты представленные жюри. В противном случае, программа участника считается некорректной и участнику сообщается причина, по которой она не была зачтена.

За каждую зачтенную задачу, участнику начисляется штрафное время равное времени, прошедшему с начала соревнования до успешной сдачи данной задачи плюс двадцать штрафных минут за каждую неправильную попытка сдать эту задачу. Штрафное время, полученное за разные задачи, суммируется. За не зачтенные задачи штрафное время не назначается.

При подсчете рейтинга, участники сначала сортируются по количеству решенных задач (по убыванию), затем, по штрафному времени (по возрастанию).

2.4.2. Всероссийская командная олимпиада школьников по программированию

Данный регламент был использован при проведении командных олимпиад школьников по программированию в 1997-1999г.

Участники могут отправлять решения на проверку и получать соответствующие результаты в течение всего соревнования.

Для тестирования каждой задачи, используется набор из десяти тестов. За правильный ответ на тест, выданный программой участника, ему начисляется один балл. Если были зачтены все 10 тестов, то ему дополнительно начисляется десять баллов.

Результатом участника по каждой задаче считается решение, набравшее наибольшее количество баллов на данной задаче.

Штрафное время начисляется так же, как на Студенческом Командном Чемпионате Мира по Программированию, но вместо зачтенного решения, используется решение, набравшее наибольшее количество баллов по этой задаче.

2.4.3. Всероссийская олимпиада школьников по информатике

Данный регламент, практически без изменений используется уже более 5 лет для проведения личных олимпиад школьников по программированию.

Решения участников собираются после окончания соревнования и тестируются в их присутствии.

Для каждой задачи применяется специально разработанный набор тестов. За прохождение каждого теста участнику начисляется определенное количество баллов. Баллы за отдельные тесты и задачи суммируются.

В итоговом рейтинге участники сортируются по убыванию количества набранных баллов.

В последний год была принята модификация, в соответствии с которой баллы за задачу не начисляются, если не пройден ни один тест из некоторого набора.

Для реализации описанных выше регламентов проведения соревнования в автоматической тестирующей системе была принята следующая схема: сначала Judge “играет” с тестируемой программой на наборе тестов, в результате чего формируются результаты проверки для всех использованных тестов. Причем, существуют два варианта “игры”: полная, — когда решение участника тестируется на всех тестах, и сокращенная — задача тестируется до первого, не пройденного теста. После чего Evaluator, по результатам отдельных тестов подсчитывает баллы, полученные участником за данное решение.

2.5. Общие требования к системам автоматического тестирования

2.5.1. Общие требование к автоматической системе тестирования для очных соревнований

К автоматической системе тестирования, применяемой на соревнованиях должна удовлетворять следующим требованиям:

- Надежность — Решение участника должно быть проверено, с занесение соответствующих результатов в таблицу результатов и сообщение их участнику. В противном случае, участник должен быть уведомлен о невозможности принять его решение на проверку.
- Удобство администрирования — При проведении соревнований реального времени несколько первых попыток по каждой задаче, обычно анализируются особенно тщательно, с целью проверить корректность проверяющей программы. Так же администратор должен иметь возможность произвести перетестирование некоторых подходов (используется, если в проверяющей программе были исправлены ошибки), или установить требуемый результат подхода в ручную.

- Предсказуемость — Среднее время тестирования решения участника должно быть хорошо предсказуемо. Если в очереди на тестирование стоит несколько решений, время тестирования последнего должно быть примерно равно сумме времен тестирования каждого, деленной на количество компьютеров используемых для запуска программ. При невозможности, протестировать решение участника в требуемый интервал времени, должно выдаваться предупреждение администратору.
- Корректность — система должна одинаково обрабатывать решения всех участников. Решения должны проверяться в порядке их поступления. Правила составления итоговой таблицы должны полностью соответствовать регламенту соревнований.
- Воспроизводимость — При перетестировании решений участников (без изменения параметров задач и проверяющих программ), результаты соревнования должны сохраняться.
- Возможность поведения одного соревнования в нескольких независимых точках одновременно, с обменом результатов между ними.

2.5.2. Общие требования к автоматической системе тестирования для интернет-соревнований

Автоматические системы тестирования для интернет-соревнований, кроме требований, описанных в предыдущем разделе должны удовлетворять нижеследующим требованиям:

- Автономность — Автоматическая проверяющая система должна работать круглосуточно, без вмешательства человека.
- Автоматическое восстановление после сбоев — При обнаружении сбоев в работе автоматическая система тестирования должна самостоятельно их устранять.
- Возможность проведения нескольких соревнований одновременно.

- Возможность соревноваться с участниками, ранее решавшими тот же набор задач (ту же задачу).

ГЛАВА 3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

В предыдущих главах были подробно описаны концепции положенные в основу ядра автоматической системы тестирования (PCMS2 Kernel). В этой главе описаны технические подробности реализации PCMS2 Kernel.

3.1. Интерфейс пользователя

С точки зрения PCMS2 Kernel все пользователи разделяются на две группы: администраторы и участники соревнований.

Единственным действием, которое может выполнить участник является отправка решения задачи на проверку. При этом тестирующая система должна либо сообщить участнику о не возможности принять его решение на проверку (на пример, так как время, отведенное на проведение соревнования, закончилось), либо протестировать решение и сообщить участнику и другим заинтересованным лицам его результат.

Роль администратора PCMS2 Kernel существенно более обширна. Администратор имеет право:

- добавлять новые задачи в архив, при этом они автоматически становятся доступными для тестирования
- добавлять языки программирования, на которых можно писать решения
- Конфигурировать общую топологию распределенного тестирующего ядра (количество используемых компьютеров и какая часть ядра, на каком из них будет запускаться)
- Приостанавливать и возобновлять тестирование некоторых решений (на пример, в связи с обнаружением ошибок в проверяющей программе)
- Повторно тестировать уже проверенные решения (возможно, с сообщением результата участнику)

Административные задачи решаются отдельным набором компонент так и называемых “административные компоненты”. Такой подход, позволяет

запускать PCMS2 Kernel, в “облегченном” режиме, когда вмешательство администратора не подразумевается (на пример, администратор редко вмешивается в работу тестирующего ядра обслуживающего интернет-соревнования), и административные компоненты не загружены.

3.2. Внутренняя архитектура PCMS2 Kernel

На основании опыта предыдущих разработок (NPC2 и APPES), при проектировании PCMS2 была выбрана компонентная модель.

При использовании данной модели программный продукт реализуется как совокупность независимых компонент, общающихся друг с другом стандартным образом по заранее оговоренным интересам. Данный подход имеет то преимущество, что различные компоненты могут быть реализованы различными группами программистов, но если соблюдены спецификации всех интерфейсов итоговая система будет успешно работать.

Вторым преимуществом компонентного подхода является простота расширения получаемого программного продукта на те области, на которые он изначально не был рассчитан. На пример, для включения в PCMS2 возможность тестировать программы под Linux, необходимо написать только одну (!) новую компоненту.

Для обеспечения требований межплатформенной переносимости (что неявно заложено в требовании обеспечить процесс тестирования решений участников под различными операционными системами), большая часть кода PCMS2 написана на Java. И при соблюдении несложных правил, достигается практически автоматическая переносимость написанного кода.

Вторым элементом переносимости является использование для файлов платформенно-независимых форматов. На пример, все конфигурационные файлы PCMS2 используют формат XML [10], который не требует накладных расходов при переносе между платформами.

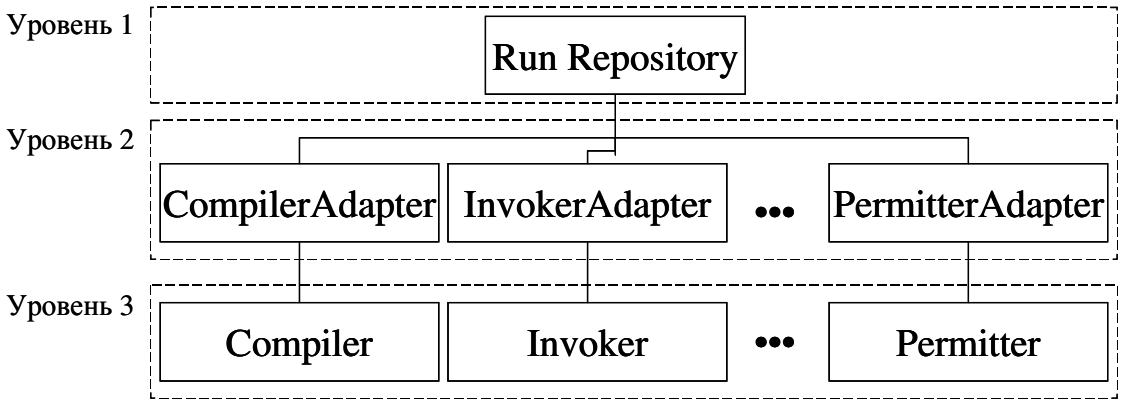

Рис 5. Трехуровневая модель тестирующего ядра.

На данный момент, единственной платформенно-зависимой частью PCMS2 является компонента, создающая “песочницу” для тестирования задач под Windows NT. Весь остальной код является полностью переносимым.

3.2.1. Трехуровневая модель тестирующего ядра

В главе 2 была приведена общая логическая схема тестирующего ядра. Данная схема является существенно двухуровневой — четко разделяется информационное ядро, представляемое RunRepository, и остальные компоненты с ним взаимодействующие. При этом остальные компоненты ядра практически не взаимодействуют друг с другом напрямую. Все общения идет только через RunRepository.

Фактически, непосредственное применение данной схемы для разбиения на ядра компоненты является неприемлемым, так как компоненте соответствующей RunRepository необходимо реализовывать большое количество не связанных между собой интерфейсов, что приводит к сильной зависимости данной компоненты от других частей тестирующего ядра, что является существенным недостатком.

Для исправления этого недостатка автором была предложена трехуровневая схема организации тестирующего ядра. Между уровнями, выделенными в предыдущей схеме, вводится промежуточный уровень “адаптеров”, которые с одной сторон, общаются с RunRepository по одному

интерфейсу, что в свою очередь позволяет существенно упростить его реализацию. С другой стороны, адаптеры предоставляют более высокоуровневый интерфейс, используемые компонентами 3-го уровня.

Вторым достоинством трехкомпонентного подхода является возможность задания порядка применения различных компонент к проверяемому решению на уровне адаптеров (на пример, очевидно, не имеет смысла вызывать Invoker, до того, как решении было обработано Compiler).

3.2.2. Общая схема взаимодействия тестирующих компонент

Тестирование решения участника происходит по следующему сценарию:

- Receiver принимает решение и помещает его в RunRepository. Если это сделать не удалось, информирует участника об ошибке.
- Permitter проверяет исходные тексты программ на наличие запрещенных конструкций (на пример, ассемблерных вставок), при нахождении которых информирует администратора. Используется только на очных соревнованиях.
- Compiler компилирует исходный текст решения участника.
- Judge “играет” с решением. Формирует запросы на запуск решения к Invoker и оценивает результаты.
- Evaluator по решению администратора, задерживает некоторые результаты тестирования. Используется только на очных соревнованиях для проверки проверяющих программ.
- Notifier сообщает участнику о результатах тестирования, модифицирует таблицу результатов для соревнований реального времени.

Переход к следующему этапу не может быть совершен, пока не завершится предыдущий. Исключением является генерирование ошибки Security Violation компонентой Permitter и ошибки Compilation Error компонентой Compiler. В этом случае, сразу производится передача управления компоненте Evaluator.

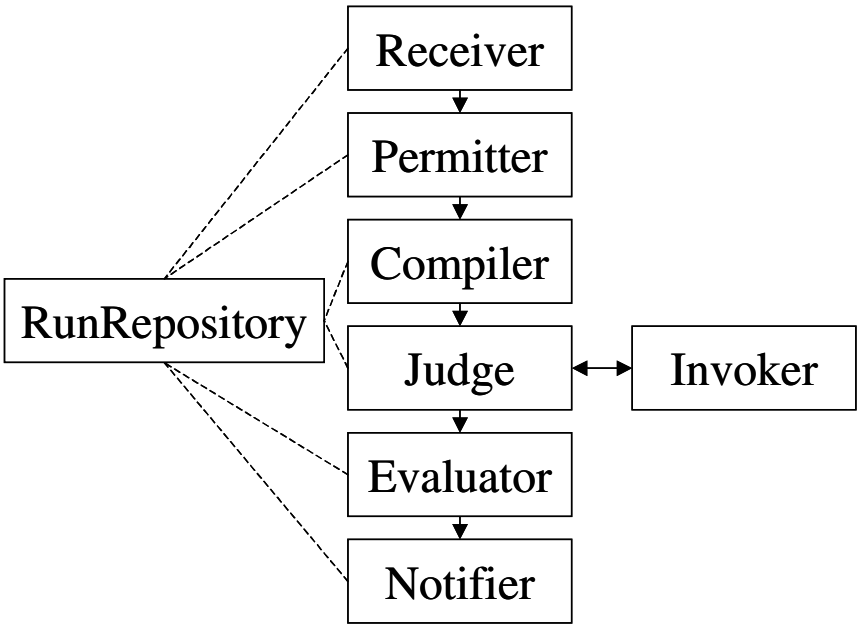

Рис. 6. Схема взаимодействия компонент тестирующего ядра.

3.3. Описание отдельных компонент PCMS2 Kernel

3.3.1. Репозиторий (RunRepository)

RunRepository является единственной компонентой, хранящей информацию о решении участников. Остальные компоненты не имеют права хранить какую-либо информацию о тестируемых решениях. Не одно действие, производимое над решением, не считает выполненным, пока его результаты не помещены в RunRepository.

Положительной чертой такого централизованного подхода является необходимость написания только одной компоненты, ответственной за сохранение важной информации между перезапусками тестирующей системы. Это позволяет сосредоточить работы по обеспечению надежности тестирующей системы в одном месте.

Так как, все остальные компоненты, по соглашению не имеют право хранить какую-либо информацию о проверяемых решениях, то их перезапуск в любой момент (на пример, в случае сбоя питания) не приводит к каким-либо негативным последствиям, кроме, может быть, повторного выполнения

небольшого объема работы (связанного с тем, что реально уже исполненные действия не были отражены в RunRepository).

При проектировании интерфейса RunRepository одним из дополнительных требований была возможность эффективной реализации данного интерфейса как на основе хранилища, использующего базу данных, так и возможность более простой реализации.

Информация, хранимая в RunRepository, была поделена на 3 непересекающиеся части.

- **Константы**, хранящие неизменную информацию, существенную для выбора способа проверки решения
- **Флаги**, отражающие текущий этап тестирования решения
- **Переменные**, хранящие дополнительную информацию, для передачи между компонентами

При работе, компонента третьего уровня формирует запрос адаптеру на выдачу следующего решения, с которым может оперировать данная компонента. Адаптер, в свою очередь, запрашивает RunRepository на наличие решений, у которых константы и флаги имеют требуемое значение. RunRepository не позволяет осуществлять выборку решений, основываясь на значениях переменных.

Такой подход позволяет эффективно реализовать RunRepository на основе базы данных.

К константам относятся следующие параметры решения:

- **Идентификатор решения (Run ID)** — Текстовая строка, уникально идентифицирующая данное решение.
- **Идентификатор задачи (Problem ID)** — текстовая строка, уникально идентифицирующая тестируемую задачу. Значение данной константе требуется Judge для определения, какие из решений, имеющихся в RunRepository, он может проверить.

- **Идентификатор языка (Language ID)** — текстовая строка, уникально идентифицирующая язык программирования, использованный для написания решения. Используется Compiler и Invoker для определения, какие из решений, имеющихся в RunRepository, он может откомпилировать и запустить соответственно.
- **Идентификатор сессии (Session ID)** — текстовая строка, уникально идентифицирующая участника сдавшего решение на тестировании и конкретный контекст, в котором это было сделано (важно, в случае, когда один участник решает одну и ту же задачу в разных соревнованиях, или решения соревнования повторно). Используется Notifier, административными компонентами и для создания списка результатов.

Очевидно, что в процессе тестирования ни один из этих параметров не может быть изменен, что и дало название данной части.

Посредством флагов осуществляется отметка того, что некоторые операции, над решением уже были выполнены. На пример, установленный флаг Compiled, означает, что компиляция решения была успешно проведена, и повторно отдавать его компоненте Compiler, не имеет смысла.

Два флага зарезервированы для обнаружения ошибок и административных нужд:

- **Active (активный)** — Данный флаг устанавливается сразу после помещения решения в RunRepository. Если он сброшен, соответствующее решение не может быть передано на обработку ни одной из не административных компонент. Сброс и установка данного флага (кроме изначальной установки), могут производить только административные компоненты.
- **Fail** (флаг ошибки) — В процессе тестирования, была обнаружена ошибка, которая не может возникнуть при нормальном функционировании автоматической тестирующей системы. На пример,

пришло решения на языке программирование, не известном ядру проверяющей системы. Данный флаг может установить любая компонента, обнаружившая ошибку. Сбросить данный флаг может только административная компонента (после устранения ошибки администратором). При установке данного флага, сообщение об ошибке должно быть записано в переменную FailComment. Решение, с установленным флагом Fail, не может быть передано на обработку компонентам, отличным от административных. Обычно, установка данного флага сигнализирует о несоответствии конфигураций различных частей автоматической системы тестирования.

С каждым флагом и переменной ассоциирован номер ревизии. При изменении значения флага (переменной), номер ревизии автоматически увеличивается. При этом если уже установленный флаг явным образом устанавливают еще раз, то соответствующий номер ревизии так же увеличивается.

При внесении изменений в RunRepository, номер ревизии всех задействованных флагов и переменных сравнивается с текущим номером ревизии. Изменения принимаются только если, все номера ревизий совпадают. Данный механизм предусмотрен для предотвращения случайного замещения данных, записанных в репозиторий новыми.

Обновление информации, содержащейся в RunRepository, происходит посредством транзакций. При этом одновременно осуществляется попытка изменения некоторого набора флагов и переменных. Транзакция или целиком принимается (если не обнаружен конфликт ревизий), либо целиком отклоняется. При этом компоненте не сообщается о принятом решении.

При внесении изменений в RunRepository все компоненты должны контролировать номер ревизии флага Fail. Таким образом, если одна из компонент установила этот флаг (и соответственно, изменила его номе ревизии), то все операции, производимые с соответствующим решением, будут

автоматически считаться не произведенными, и новая информация после установки данного флага в RunRepository заноситься не будет. Таким образом, после сброса флага Fail (административной компонентой), решение будет находиться точности том же состоянии, которое вызвало ошибку, и ошибочное действие будет повторено (подразумевается, что администратор нашел и устранил причину ошибки).

3.3.2. Адаптеры

Каждой компоненте 3-го уровня соответствует свой адаптер, который преобразует запросы высокуровнего интерфейса компонента в низкоуровневый интерфейс RunRepository

При использовании высокоуровнего интерфейса не используются таки понятия, как флаги и переменные, наоборот, операции производятся на уровне сложных структур данных. Задаче адаптера является упаковка их запись этих структур в RunRepository.

Так же на адаптер возложено контролирования правильного состояния флагов Active и Fail.

Обычно, адаптер экспортирует метод с**onnect**, создающий новый новое соединение с RunRepository, используя переданные параметры. При этом формируется объект типа IConnection, имеющий следующие методы:

- **Get** — Выдает новое задание на обработку решения, удовлетворяющего требуемым параметрам.
- **Done** — Помещает в RunRepository информацию об обработке решения. При этом производится анализ переданного результата, и формируются новые значения флагов и переменных.
- **Close** — Закрывает соединение и освобождает связанные с ним ресурсы.

Рассмотрим взаимодействие адаптера и соответствующей компоненты на примере связки Compiler — CompilerAdapter.

Compiler использует высокоуровневый интерфейс CompilerService:

```
/**
 * High-level compiler-to-repository interface.
 *
 * @author Georgiy Korneev
 * @version $Id: CompilerService.java,v 1.8 2002/02/26 13:56:53$
 */
public interface CompilerService extends Service {
    /** Opens a new connection. */
    public IConnection connect(String[] laguages)
        throws RemoteException;

    /** Provides compiler with requests. */
    public static interface IConnection {
        /** Gets a new request. */
        public IRequest get();

        /** Informs provider that request has been processed. */
        public void done(Result result)

        /** Closes the connection. */
        public void close();
    }

    /** Provides information needed to compile the run. */
    public static interface IRequest {
        /** Gets the run ID. */
        public String getRunId();

        /** Gets the language ID of sources. */
        public String getLanguageId();

        /** Gets VFL of the sources. */
        public VFL getSources() throws RemoteException;
    }
```

```
    /** Represents compilation result. */
    public static class Result {
        /** ID of the compiled run. */
        public String run_id;

        /** <code>true</code> if compiler has properly processed
         * request, <code>false</code> otherwise. */
        public boolean success;

        /** <code>true</code> if compilation
         * succeded (no compilation error
         * occured), <code>false</code> otherwise. */
        public boolean compiled;

        /** Human readable comment. */
        public String comment;

        /** VFL of binary executables. */
        public VFL executables;
    }
}
```

Compiler инициирует создания нового соединения с RunRepository, вызывая метод `CompilerService.connect`. В качестве параметров данному методу передается список идентификаторов языков программирования, которые данный компилятор может обработать. CompilerAdapter создает низкоуровневое соединение с RunRepository, указывая в качестве параметров: флаги Active и Permitted установлены, а флаги Fail, Compiled и HasResult сброшены, при этом идентификатор языка решения, должен входить в переданный список. Флаги Active и Fail проверяются в соответствии со стандартным соглашением. Флаг Permitted проверяется, что бы убедиться, что Permitter уже обработал данного

решения, а Compiled — еще не откомпилировано, HasResult — что результат тестирования решения еще не известен.

Для компилирования решения, необходимы следующие данные:

- **Идентификатор решения (Run ID)** — Необходим для последующего указания, какое решение было откомпилировано.
- **Идентификатор языка (Language ID)** — Для определения точной процедуры компиляции.
- **Файлы с исходными текстами (Sources)** — Собственно, исходные тексты решения, подлежащие компиляции.

Идентификаторы решения и языка являются константными параметрами решения, а файлы с исходными текстами — переменной с соответствующим названием (Sources).

Соответственно, к запросу на выборку информации из RunRepository добавляется запрос на переменную Sources.

В результате компиляции решения формируются значения следующих переменных:

- **Комментарий компилятора (CompilerComment)** — строка, предназначенная для краткого описания результат компиляции (в частности, выявленных ошибок).
- **Исполняемые файлы (Executables)** — набор исполняемых файлов, полученный в результате компиляции.

И флагов:

- **Флаг ошибки (Fail)** — устанавливается, если решение не удалось откомпилировать из-за административной ошибки (в результате, выданном Compiler, `success` установлен в `false`).
- **Флаг компиляции (Compiled)** — Устанавливается, если компиляция успешно завершена (в результате, выданном Compiler, `success` и `compiled` установлены в `true`).

- **Флаг окончания тестирования (HasResult)** — Устанавливается, если произошла ошибка компиляции, т.е. исходные тексты не корректны (в результате, выданном Compiler, `success` установлен в `true`, а `compiled` установлены в `false`).

Таким образом, CompilerAdapter должен запросить номера ревизий соответствующих переменных, что бы включить их в запрос на выполнения. Модификация запроса на флаги не требуется, так как их значения уже затребованы.

Из вышесказанного следует, что при работе CompilerAdapter запрашивает у RunRepository решения, удовлетворяющие маске (список языков программирования, Active, Permitted, !Fail, !Compiled, !HasResult), и соответствующие им переменные Sources, CompilerComment, Executables. Где восклицательные знаки предшествуют флагам, которые должны быть сброшены. После компиляции CompilerAdapter модифицирует один из флагов Fail, Compiled, HasResult и устанавливает значения переменных CompilerComment и Executables.

3.3.3. Компилятор (Compiler)

Основной задачей компоненты Compiler является компиляция решений участников. Compiler получает на входе идентификатор языка программирования на котором написано решение и исходные тексты решения. На выходе Compiler должен выдать либо сообщение об ошибке компиляции либо набор исполняемых файлов. Все операции компилятор осуществляет через интерфейсы CompilerService, предоставляемый CompilerAdapter и NativeInvokeService, предоставляемый NativeInvoke.

Компонента Compiler может одновременно компилировать несколько решений, распределяя задания по различным NativeInvoke. При распределении заданий на выполнение проверяется, может ли конкретный экземпляр NativeInvoke выполнить требуемое задание. Если сформированное задание не

один из компонентов NativeInvoke выполнить не может, Compiler сигнализирует об ошибке, устанавливая флаг Fail.

Конфигурация компилятора состоит из списка языков, поддерживаемых данным компилятором и правил компилирования для каждого из них.

Рассмотрим конфигурацию языка программирования на примере Borland C++ 3.1:

```
<language
    id              = "cpp.borland.3.1"
    compile-files   = "{sources#.*\.cpp$}"
    compile         = "bcc {current}"
    link            = "bcc -ml {sources#.*\.cpp$:!.obj}"
    binaries        = "{main:!.exe} "
    tools-exec-id   = "bin.x86.exe.dos"
/>
```

Данная конфигурация содержит следующие атрибуты:

- `id` — Идентификатор языка программирования. По соглашению, части идентификатора записываются через точку, последовательно уточняя конкретный язык
- `compile-files` — маска файлов, которые к которой будет применена команда `compile`. В данном случае, выделяются файлы с расширением `.cpp`
- `compile` — команда, применяемая по очереди, к каждому файлу, попадающего под маску compile-files.
- `link` — команда, исполняемая после того, как все файлы будут откомпилированы. В данном случае, в команду `bcc -ml` подставляется список файлов с расширением `.cpp`, предварительно замененным на `.obj`
- `binaries` — маска файлов, необходимых для исполнения откомпилированного решения.

- `tools-exec-id` — идентификатор типа исполняемых файлов, используемых при компиляции.

Рассмотрим подробнее формат команд и масок, используемых для описания процесса:

```
Command     ::= ([^{] | expression)*
file-list   ::= ' '* (filename | expression) [' '+ (filename |
                expression)]*
expression  ::= '{' file-group [':' pattern]'}'
file-group  ::= type ['#' filter]
Type        ::= 'sources' | 'main' | 'current' | 'all' | 'exec' |
                'execs' | 'input' | 'inputs' | 'output' | 'outputs'
                | 'answer' | 'answers'
Filter      ::= [^}:#]+
Pattern     ::= [^}:#]+
Filename    ::= [^ ]+
```

Используемые типы групп файлов (так же перечислены группы файлов, используемые в других разделах):

Наименование	**Описание**
`Sources`	Исходные файлы решения
`Main`	Первый исходный файл
`current`	Текущий исходный файл
`All`	Все файлы в текущем каталоге

При применении шаблонов используются следующие правила:

Если опущен шаблон (`pattern`) используется `!.!`

Если опущен фильтр (`filter`) используется `.*`

Шаблон “`!`” заменяется именем файла без расширения (для каждого файла в группе)

Шаблон “`!.!`” заменяется именем файла с расширением (для каждого файла в группе)

Шаблон “`!/`” заменяется полным путем к файлу (для каждого файла в группе)

“`{file-group#filter:mask}`” заменяется списком именем файлов, разделенных пробелами. Используются файлы из `file-group`, удовлетворяющие фильтру и модифицированный в соответствии с шаблоном.

Таким образом, описывается процесс компиляции всех современных языков программирования.

3.3.4. Судья (Judge)

Компонента Judge “играет” с решением участника, и определяет предварительный результат тестирования.

При использовании тестирования на наборе тестов, Judge просто формирует задания для Invoker и проверяет полученные результаты с помощью проверяющей программы.

Правила формирования запроса на исполнение решения участника и последующей проверки написаны в файле конфигурации задачи, который Judge получает от ProblemArchive.

Рассмотрим конфигурацию задачи на конкретном примере:

```
<problem id = "acm.neerc.2001.qf.southern.a">
  <testers>
    <acm-tester
        tests   = "6"
        inputs  = "tests/{test##}"
        answers = "tests/{test##}.a"

        input-names   = "input.txt"
        output-names  = "output.txt"

        time-limit    = "1s"
        memory-limit  = "1m"
    >
```

```
        <checker
            class   = "pcms2.testlib.Checker"
            program = "ncmp.exe"
            command = "{exec:!/!.!} {input} {output} {answer}
                       result.xml -appes"
            exec-id = "bin.x86.exe.win32"
            result  = "result.xml"
        />
      </acm-tester>
    </testers>
</problem>
```

Идентификатор задачи (`id`) указывается в основном тэге конфигурации задачи (`<problem>`).

Вложенная структура `<testers>` описывает возможные методы тестирования данной задачи.

В примере определен единственный метод тестирования, по регламенту ACM ICPC (вложенный тэг `<acm-tester>`). В общем случае, для задачи может быть определено произвольное количество методов тестирования.

В методе тестирования обычно задаются следующие параметры:

- `tests` — Количество тестов
- `inputs` — Шаблон имен входных файлов в архиве
- `answers` — Шаблон имен файлов ответов
- `input-names` — Имена входных файлов
- `output-names` — Имена выходных файлов
- `time-limit` — Максимальное время работы решения на одном тесте
- `memory-limit` — Максимальный объем памяти, который может затребовать решение.

Для метода тестирования указывается используемая проверяющая программа (`<checker>`). В данном примере используется внешняя проверяющая программа `ncmp.exe`.

3.3.5. Evaluator

Производит окончательную оценку результатов полученных Judge. Используется на соревнованиях для ручного подтверждения первых результатов проверки, с целью контроля правильности проверяющих программ.

3.3.6. Исполнитель (Invoker)

Invoker подготавливает каталог, для запуска в нем решения пользователя и анализирует файлы, созданные в процессе запуска.

Одной из задач Invoker является копирование фалов с входными данными в рабочий каталог решения и получение файлов с выходными данными.

Так же Invoker точно специфицирует параметры песочницы для NativeInvoke и проверяет рабочий каталог на наличие посторонних файлов.

3.3.7. NativeInvoke

Фактически является платформенно-независимой библиотекой, для запуска программ, в том числе с созданием защищенной “песочницы”, позволяющей прерывать некорректные программы участников.

При запуске экспортирует список поддерживаемых форматов исполняемых файлов. Компоненты, которым требуется исполнить какую-либо программу, обращаются к NativeInvoke с соответствующим запросом.

ЗАКЛЮЧЕНИЕ

Тестирующая система PCMS2 Kernel была успешно реализована и апробирована на реальных соревнованиях.

На данный момент она успешно работает как часть проекта Интернет-Школы Программирования, осуществляя автоматическое тестирование знаний учащихся.

В рамках данной тестирующей системе была успешно реализована и апробирована игровая стратегия тестирования, которая позволила расширить множество потенциально тестируемых задач.

На основе PCMS2 Kernel в 2001 году были проведены:

- Четвертьфинал Командного студенческого чемпионата мира по программированию.
- Вторая Всероссийская командная олимпиада школьников по программированию
- Полуфинал Командного студенческого чемпионата мира по программированию

PCMS2 применялась для подготовки студентов СПбГИТМО (ТУ) к соревнованиям по программированию. На ее базе осуществлялось тестирование абитуриентов и студентов кафедры Компьютерных Технологий.

PCMS2 Kernel круглосуточно поддерживает систему автоматического тестирования интернет-школы программирования [9] и систему автоматического проведения online-соревнований [8].

За время функционирования PCMS2 Kernel было проверено более 10000 решений.

В дальнейшем планируется разработка компонент, позволяющих тестировать программы для операционной системы Linux и ей подобных.

СПИСОК ЛИТЕРАТУРЫ

Публикации

1. Корнеев Г.А. Автоматизированная система тестирования программ. // Материалы VIII международной конференции "Современные технологии обучения <<СТО-2002>>". 24 апреля 2002 года. -- Том 2, с.327-329. -- СПб.: СПбГЭТУ, 2002.
2. Отчет о НИР №20025 “Создание информационной обучающей системы подготовки разработчиков программного обеспечения на основе методологии, используемой в международных олимпиадах по программированию” (код проекта 3329) (за 2000 год).
3. Отчет о НИР №20069 “Разработка концепции и методического обеспечения новой технологии обучения и контроля качества образования в области программирования” (заключительный за 2001 год)

Ресурсы глобальной сети Интернет

4. http://icpc.baylor.edu — официальный сайт Командного студенческого чемпионата мира по программированию
5. http://www.ioi2001.edu.fi — сайт Международной олимпиады школьников по программированию 2001 года
6. http://acm.uva.es — сайт в испанском городе Вальядолиде — первый архив задач с автоматической системой проверки
7. http://acm.timus.ru — сайт в городе Екатеринбурге — первый российский архив задач с автоматической системой проверки
8. http://neerc.ifmo.ru/online — архив соревнований с системой автоматического проведения online-соревнований
9. http://ips.ifmo.ru — Интернет-школа Программирования
10. http://www.w3.org/TR/2000/REC-xml-20001006 — W3C XML 1.0 specification

11. Казаков М.А. Разработка и внедрение системы поддерживающей новые технологии обучения программированию

ПРИЛОЖЕНИЯ

3.4. Приложение 1. Конфигурационные файлы компонент тестирующего ядра

3.4.1. Compiler

```
<compiler
    start   = "always"
    threads = "2"
>
    <language
        id              = "pascal.borland.7.0"
        link            = "tpc.exe -m {main:!.!}"
        binaries        = "{main:!.exe}"
        tools-exec-id   = "bin.x86.exe.dos"
    />

    <language
        id              = "delphi.6.0"
        link            = "dcc32.exe -cc {main:!.!}"
        binaries        = "{main:!.exe}"
        tools-exec-id   = "bin.x86.exe.win32"
    />

    <language
        id              = "cpp.borland.3.1"
        compile-files   = "{sources#.*\.cpp$}"
        link            = "bcc -ml {main:!.!}"
        binaries        = "{main:!.exe} {sources:!.obj}"
        tools-exec-id   = "bin.x86.exe.dos"
    />

    <language
```

```
    id              = "cpp.borland.3.1-batch"
    compile-files   = "{sources#.*\.cpp$}"
    compile         = "bcc -ml -c {current:!.!}"
    link            = "bcc -ml {sources:!.obj}"
    binaries        = "{main:!.exe} "
    tools-exec-id   = "bin.x86.exe.dos"
/>

<language
    id              = "c.borland.3.1"
    compile-files   = "{sources#.*\.c$}"
    link            = "bcc.exe -ml {main:!.!}"
    binaries        = "{main:!.exe} {sources:!.obj}"
    tools-exec-id   = "bin.x86.exe.dos"
/>

<language
    id              = "basic.microsoft.qb.45"
    compile-files   = "{sources#.*\.bas$}"
    compile         = "qbComp {current:!.!}
                       /O/Ot/G2/Fs/Lr/FPi/T/C:512;"
    link            = "qbLink
                       /EX {sources:!.obj}, {main:!.exe},
                       NUL,c:\LANGUAGE\QB\BCL71EFR.LIB;"
    binaries        = "{main:!.exe}"
    tools-exec-id   = "bin.x86.exe.dos"
/>

<language
    id              = "java.sun.1.3.2"
    link            = "java -jar l:\JavaCompiler.jar
                       temp {main:!.jar} {main:!.!}"
    binaries        = "{main:!.jar}"
    tools-exec-id   = "bin.x86.exe.win32"
```

```
    />
</compiler>
```

3.4.2. Invoker

```
<invoker
    start       = "always"
>
    <thread
       work-dir    = "i:/1/work/"
       bin-dir     = "i:/1/bin/"
    />
    <thread
       work-dir    = "i:/2/work/"
       bin-dir     = "i:/2/bin/"
    />

    <language
        id              = "pascal.borland.7.0"
        executable-id   = "bin.x86.exe.dos"
    />
    <language
        id              = "delphi.6.0"
        executable-id   = "bin.x86.exe.win32"
    />
    <language
        id              = "c.borland.3.1"
        executable-id   = "bin.x86.exe.dos"
    />
    <language
        id              = "cpp.borland.3.1"
        executable-id   = "bin.x86.exe.dos"
    />
    <language
        id              = "basic.microsoft.qb.45"
```

```
        executable-id   = "bin.x86.exe.dos"
    />
    <language
        id              = "java.sun.1.3.2"
        executable-id   = "bin.java.1.3"
    />

    <executable
        id          = "bin.x86.exe.dos"
        command     = "{exec:!/!.!}"
    />
    <executable
        id          = "bin.x86.exe.win32"
        command     = "{exec:!/!.!}"
    />
    <executable
        id          = "bin.java.1.3"
        command     = "java -jar {exec:!/!.!}"
    />
</invoker>
```

3.5. Приложение 2. Пример конфигурации задач

```
<problem
    id = "acm.neerc.2000.sf.a"
>
    <testers>
        <acm-tester
            tests   = "20"
            inputs  = "tests/{test##}"
            answers = "tests/{test##}.a"

            input-names   = "flip.in"
            output-names  = "flip.out"
```

```
                time-limit    = "10s"
            >
                <checker
                    class   = "pcms2.testlib.Checker"
                    program = "check.exe"
                    command   =   "{exec:!/!.!}   {input}   {output}
{answer} result.xml -appes"
                    exec-id = "bin.x86.exe.win32"
                    result  = "result.xml"
                />
            </acm-tester>
        </testers>
    </problem>
```

3.6. Приложение 3. Пример проверяющей программы

```
{ VERIFICATION PROGRAM for FLIP GAME problem for NEERC'2000 }
{$A+,B-,D+,E+,F-,G-,I+,L+,N+,O-,P-,Q+,R+,S+,T-,V+,X+,Y+}
program FLIP_CHECK;
uses
  testlib, symbols;

var
  iouf, ians: longint;

function readAns(var stream: InStream): longint;
var
  s: string;
  i: longint;
  r: integer;
begin
  s := compress(stream.readString);
  if upstr(s) = 'IMPOSSIBLE' then
    readAns := -1
  else begin
```

```
    val(s, i, r);
    if r <> 0 then
      stream.Quit(_PE, 'Invalid answer string: ' + s);
    if i < 0 then
      stream.Quit(_PE, 'Negative number in answer: ' + s);
    readAns := i;
  end;
end;

begin
  ians:= readAns(ans);
  iouf:= readAns(ouf);
  if not ouf.seekeof then Quit(_PE, 'Extra data in file');
  if iouf <> ians then
    Quit(_WA, 'Wrong answer: ' + str(iouf,0) +
    ' <> ' + str(ians,0) +
      ' (-1 means impossible)');
  Quit(_OK, 'Ok');
end.
```